

Blog

Tecnología para Desarrollo

Tecnología para Desarrollo

Tecnología para Negocio

Eventos y seminarios

Trazabilidad Distribuida con Spring Cloud: Sleuth y Zipkin

por [Eduardo González](https://www.paradigmadigital.com/author/egonzalez/) (<https://www.paradigmadigital.com/author/egonzalez/>)

Madrid, España

12 de enero del 2017

Full Cloud

6 comentarios

El auge de las [arquitecturas de microservicios](#)

(<https://www.youtube.com/watch?v=LjqtaaMJi4U>) ha traído consigo algunos retos que debemos ser capaces de abordar para conseguir un sistema consistente, como son la monitorización, gestión de la configuración centralizada, centralización de logs...

En este tipo de arquitecturas una petición de un consumidor de nuestro sistema puede desencadenar varias llamadas internas entre microservicios, por lo que es importante

This website uses a cookie to improve your user experience. We'll assume you're ok with this, but you can opt-out if you want to. [Accept](#) [Read More](#)

posteriormente consultar las peticiones de forma centralizada. Spring pone a nuestra disposición herramientas que nos facilitan este trabajo: [Spring Cloud Sleuth](https://cloud.spring.io/spring-cloud-sleuth/) (<https://cloud.spring.io/spring-cloud-sleuth/>) y [Zipkin](https://github.com/openzipkin/zipkin) (<https://github.com/openzipkin/zipkin>).

En este post explicaremos la forma de almacenar y explotar de forma centralizada la trazabilidad de peticiones a nuestros servicios utilizando estas herramientas.



(https://www.paradigmadigital.com/wp-content/uploads/2016/11/shutterstock_97887752_opt.jpg)

Spring Cloud Sleuth

La primera pieza de la ecuación para conseguir identificar inequívocamente una petición es [Spring Cloud Sleuth](https://cloud.spring.io/spring-cloud-sleuth/) (<https://cloud.spring.io/spring-cloud-sleuth/>). Es una librería que implementa una solución de trazado distribuido para Spring Cloud. Con tan solo incluir la dependencia con Sleuth en los microservicios, dotamos al ecosistema de un mecanismo automático de identificación de peticiones, ya que añade varios campos útiles a las mismas para identificarlas.

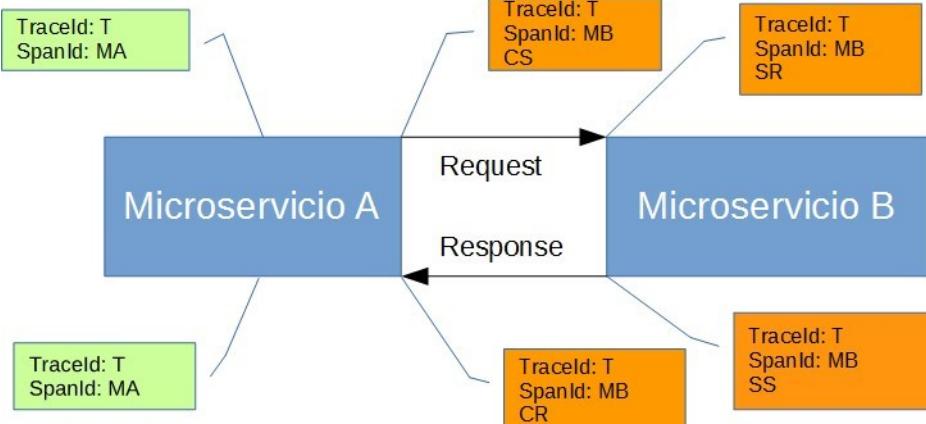
This website uses cookies to improve your experience. We'll assume you're ok with this, but you can opt-out if you wish. [Accept](#) [Read More](#)

spanId (son los que realmente aseguran una traza única), los cuales incluye en las salida de log del microservicio. El sistema de trazado utilizado por Sleuth se basa en terminología Dapper, cuyos principales conceptos son:

- › **Trace**: Un conjunto de spans que forman una estructura de árbol de llamadas, forma la traza de la petición.
- › **Span**: Es la unidad básica de trabajo, por ejemplo una llamada a un servicio. Se identifican con un *id de span* y un *id de trace* a la que pertenece dicho span. Tienen inicio y fin, y con ello se consigue trackear el tiempo de respuesta entre peticiones.
- › **Annotation**: Se usa para grabar un evento en el tiempo. Las anotaciones más importantes que usa internamente son:
 - › **cs** (Client Sent): El cliente ha hecho una petición. Inicio del span.
 - › **sr** (Server Received): El servidor ha recibido la petición y empieza su procesado. $timestamp_{sr} - timestamp_{cs} = latencia\ de\ red$.
 - › **ss** (Server Sent): Envío a cliente desde el servidor de la respuesta. $timestamp_{ss} - timestamp_{sr} = tiempo\ de\ procesamiento\ de\ petición\ en\ servidor$.
 - › **cr** (Client Received): Fin del span. El cliente ha recibido correctamente la respuesta del servidor. $timestamp_{cs} - timestamp_{cr} = tiempo\ total\ de\ la\ petición$.
- › **Tag**: Par clave/valor que identifica cierta información en el span. No contiene timestamps, simplemente identifica.

A continuación se muestra un diagrama de cómo se

Todos los datos se acopian en una sola traza entre dos microservicios que hacen uso de Sleuth. [Accept](#) [Read More](#)



(<https://www.paradigmadigital.com/wp-content/uploads/2016/11/td1.jpg>)

Además de generar los identificadores únicos y añadirlos a los logs de la aplicación, es necesario que se propaguen de forma correcta entre los microservicios que forman parte de la petición.

Para ello incluye una capa que gestiona automáticamente dicha propagación de los datos de trazado de Sleuth entre los sistemas más comunes de comunicación entre microservicios Spring Cloud, como son los RestTemplate, servlets, filtros Zuul, clientes feign, mensajería...

A continuación se presentan las posibilidades de implementación de la trazabilidad que ofrece Sleuth.

Implementaciones Spring Cloud Sleuth

[Spring initializr \(https://start.spring.io/\)](https://start.spring.io/) es una herramienta que Spring ha puesto a nuestra disposición para agilizar la creación de aplicaciones Spring Boot. Según la misma, existen dos posibles dependencias Sleuth:

SPRING INITIALIZR bootstrap your application now

Generate a with Spring Boot

Project Metadata

This website uses cookies to improve your experience. We'll assume you're ok with this, but you can opt-out if you wish. [Accept](#) [Read More](#)

Dependencies

Search for dependencies

Sleuth

Distributed tracing via logs with spring-cloud-sleuth



(<https://www.paradigmadigital.com/wp-content/uploads/2016/11/td2.jpg>)

- **Sleuth:** Dota a la aplicación de correlación de trazas de log entre distintos microservicios mediante cabeceras HTTP. Para tenerlo disponible, se debe incluir la siguiente dependencia en el proyecto:

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-sleuth</artifactId>
</dependency>
```

Una vez incluida se lanza una petición con un sistema de llamadas compatible con Sleuth, como puede ser

RestTemplate:

```
ResponseEntity<String> response = restTemplate.getForEntity("http://localhost:8089/getName", String.class);
```

Por cada petición que trace contendrá identificadores para definirla únicamente, como se ve en la imagen (**5501a4bcd8faaa8** el traceld y **44d49402e8b56e98** el spanId del servicio llamante):

```
2016-11-15 15:12:25.977 INFO [sleuth-client,5501a4bcd8faaa8,44d49402e8b56e98,false] 9544 --- [nio-8999-exec-4] c.p.microservices.SleuthApplication : Handling home
```

(<https://www.paradigmadigital.com/wp-content/uploads/2017/01/tra.png>)

Como se ha comentado anteriormente, estos identificadores se envían de un servicio a otro por cabeceras HTTP:
This website uses cookies to improve your experience. We'll assume you're ok with this, but you can opt-out if

you wish. [Accept](#) [Read More](#)

```

▲ 🔎 "httpRequest.getHeaderNames()= NamesEnumerator (id=142)
  ▷ f headers= MimeHeaders (id=126)
    ▷ □ next= "accept" (id=144)
    ▷ □ pos= 1
    ▷ f size= 9

==== MimeHeaders ====
accept = text/plain, application/json, application/*+json, /*
x-b3-traceid = 5501a4bacd8faaa8
x-b3-spanid = 1a424e54400f4782
x-b3-sampled = 0
x-span-name = http:/getName
x-b3-parentspanid = 44d49402e8b56e98
user-agent = Java/1.8.0_73
host = localhost:8089
connection = keep-alive

```

[\(https://www.paradigmadigital.com/wp-content/uploads/2017/01/3.png\)](https://www.paradigmadigital.com/wp-content/uploads/2017/01/3.png)

- **Sleuth Stream:** Además de dotar de correlación de mensajes, incluye las trazas en un binder de Spring Cloud Stream para su transmisión. Existen varias posibilidades de broker sobre el que enviar las trazas, **Kafka**, **Redis** o **RabbitMQ**. Para disponibilizarlo en la aplicación, se debe incluir la dependencia:

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-sleuth-stream</artifactId>
</dependency>
```

Además de la dependencia con el broker, en este caso RabbitMQ:

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-stream-rabbit</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-aop</artifactId>
</dependency>
```

A continuación se define el canal de comunicación tanto en el emisor como en el receptor:
 This website uses cookies to improve your experience. We'll assume you're ok with this, but you can opt-out if you wish. [Accept](#) [Read More](#)

```

spring:
  cloud:
    stream:
      bindings:
        sleuth-stream-save:
          destination: sleuth-stream-save
          group: sleuth-stream
          content-type: application/json

```

› Emisor

```

public interface SleuthStreamProcessor {
    String SLEUTH_SAVE = "sleuth-stream-save";

    @Output(SleuthStreamProcessor.SLEUTH_SAVE)
    MessageChannel sleuthSave();
}

```

› Receptor

```

public interface SleuthStreamProcessor {
    String SLEUTH_SAVE = "sleuth-stream-save";

    @Input(SleuthStreamProcessor.SLEUTH_SAVE)
    SubscribableChannel sleuthSave();
}

```

Y como se ve en las imágenes, el traceld y el spanId se propaga en la cabecera del mensaje enviado a RabbitMQ:

```
2016-11-16 10:28:51.258 INFO [,8fe7a7ab27d673eb,a8e2988e7795b880,false] 27564 --- [nio-8555-exec-7] c.p.microservices.SleuthController : Handling home
```

[\(https://www.paradigmadigital.com/wp-content/uploads/2017/01/traz2.png\)](https://www.paradigmadigital.com/wp-content/uploads/2017/01/traz2.png)



This website uses cookies to improve your experience. We'll assume you're ok with this, but you can opt-out if you wish. [Accept](#) [Read More](#)

Zipkin

La segunda pieza de la ecuación es [Zipkin](#) (<https://github.com/openzipkin/zipkin>). Si con Sleuth se consigue identificar los saltos entre microservicios de una petición en concreto, con Zipkin se consigue un sistema **completo** de trazabilidad distribuida para microservicios Spring Cloud: generación de trazas con *ids únicos*, almacenamiento de las mismas y posterior visualización en una interfaz gráfica para su estudio. Abraza la funcionalidad de Spring Cloud Sleuth y la amplía para formar un sistema completo de monitorización de peticiones entre microservicios.

Su objetivo es consultar la salud del ecosistema. Si se quiere tener un procesado a más alto nivel, se debe utilizar en conjunto con algún sistema de agregación de logs como Kibana o Splunk ya que estos sistemas permiten explotar las trazas de forma mucho más intensiva y personalizada.

Implementaciones Zipkin

Como Zipkin se apoya en Sleuth, tiene dos posibles sistemas de envío de trazas al servidor, **mediante cabeceras HTTP** y con **brokers de mensajería**. Según [Spring Initializr](#) (<https://start.spring.io/>) estas son las dependencias publicadas para Zipkin:

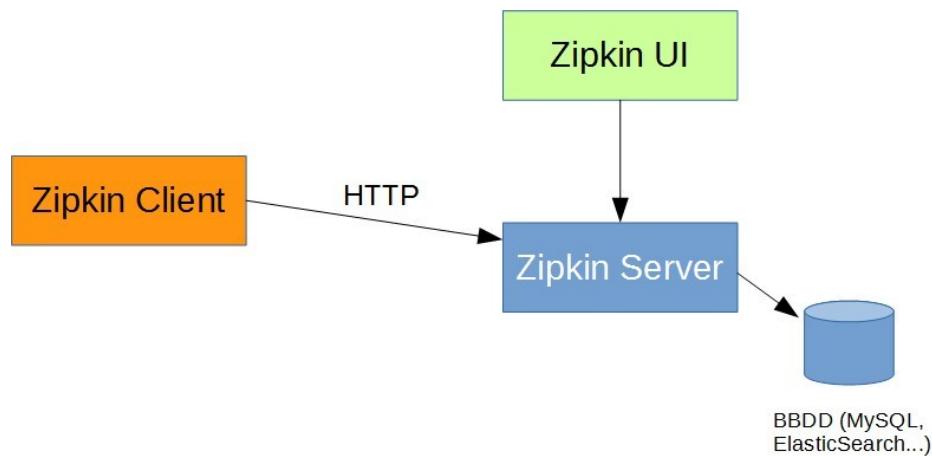
The screenshot shows the Spring Initializr interface with the following configuration:

- Project Metadata:**
 - Group: com.example
 - Artifact: demo
- Dependencies:**
 - Search for dependencies: zipkin Client
 - Zipkin Client (highlighted): Distributed tracing with an existing Zipkin installation and spring-cloud-sleuth-zipkin. Alternatively, consider Sleuth Stream.
 - Zipkin Stream: Consumes span data in messages from Spring Cloud Sleuth Stream and writes them to a Zipkin store.
 - Zipkin UI: add the Zipkin UI module to the Zipkin server to get a Zipkin service that accepts Spans and provides visualization.
- Cookie Consent Banner:** This website uses cookies to improve your experience. We'll assume you're ok with this, but you can opt-out if you wish. [Accept](#) [Read More](#)

Zipkin Server
Consumes span data over HTTP and writes them to a span store

(<https://www.paradigmadigital.com/wp-content/uploads/2016/11/td5.jpg>)

- › **Cabeceras HTTP:** Si se quiere seguir una aproximación con cabeceras HTTP, la estructura de dependencias es la siguiente:



(<https://www.paradigmadigital.com/wp-content/uploads/2016/11/td6.jpg>)

En los **microservicios proveedores de trazas**, se debe incluir la dependencia con el **Zipkin Client** (*spring-cloud-starter-zipkin*) el cual tiene embebida la dependencia con *spring-cloud-sleuth-zipkin* que compatibiliza las trazas Sleuth con las que espera recibir Zipkin.

```

<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-zipkin</artifactId>
</dependency>
  
```

Una vez generadas las trazas, se envían en la cabecera HTTP al **Zipkin Server** (*zipkin-server*).

```

<dependency>
    <groupId>io.zipkin.java</groupId>
    <artifactId>zipkin-server</artifactId>
</dependency>
  
```

This website uses cookies to improve your experience. We'll assume you're ok with this, but you can opt-out if you wish. [Accept](#) [Read More](#)

Si no se le especifica ninguna forma de almacenarlas, las deja en memoria. Esta es una mala práctica si se quiere implantar este sistema de trazabilidad en producción, por lo que se debería especificar una base de datos, como puede ser Elasticsearch (*zipkin-autoconfigure-storage-elasticsearch*).

```
<dependency>
    <groupId>io.zipkin.java</groupId>
    <artifactId>zipkin-autoconfigure-storage-
elasticsearch</artifactId>
    <version>1.2.1</version>
</dependency>
```

Se deben configurar las propiedades de conexión en el zipkin-server:

```
zipkin:
  storage:
    type: elasticsearch
    elasticsearch:
      cluster: elasticsearch
      hosts: localhost:9300
      index: zipkin
      index-shards: 5
      index-relicas: 1
```

Zipkin crea índices por fecha en Elasticsearch para mejorar su gestión:

```
C:\programas\curl-7.49.1-win32-mingw\bin>curl -X GET http://localhost:9200/_cat/indices?v
health status index  pri  rep docs.count docs.deleted store.size pri.store.size
yellow open   zipkin-2016-07-11  5    1       108          0   256.7kb   256.7kb
yellow open   zipkin-2016-07-08  5    1       129          0   268.3kb   268.3kb
yellow open   zipkin              5    1        0          0     800b     800b
```

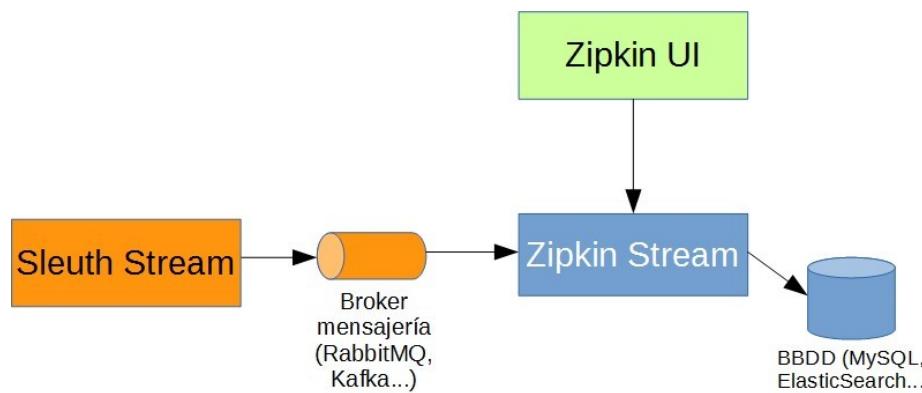
[\(https://www.paradigmadigital.com/wp-content/uploads/2016/12/traz.jpg\)](https://www.paradigmadigital.com/wp-content/uploads/2016/12/traz.jpg)

Para gestionar el volumen de información que se almacena en elastic, el creador recomienda eliminar periódicamente estos índices con [Elastic Curator \(https://www.elastic.co/guide/en/elasticsearch/client/curator/current\)](https://www.elastic.co/guide/en/elasticsearch/client/curator/current)

This website uses cookies to improve your experience. We'll assume you're ok with this, but you can opt-out if you wish. [Accept](#) [Read More](#)

Para que una aplicación Spring Boot se convierta en un Zipkin Server tan solo hace falta incluir la anotación `@EnableZipkinServer`.

- **Brokers de mensajería:** Si por el contrario se desea enviar las trazas al servidor mediante un binder de Spring Cloud Stream el sistema se conforma de la siguiente manera:



(<https://www.paradigmadigital.com/wp-content/uploads/2016/11/td7.jpg>)

Los microservicios de los que se desea obtener las trazas deben contener la dependencia con **Sleuth Stream** el cual enviará las mismas al broker configurado.

Por su parte, el **Zipkin Stream** (*spring-cloud-sleuth-zipkin-stream*) actúa de la misma forma que el Zipkin Server, pero recogiendo las trazas del broker (se debe incluir la dependencia con el mismo) y almacenándolas en memoria si no se especifica nada o en una base de datos si se especifica.

```

<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-sleuth-zipkin-
stream</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-stream-
rabit</artifactId>
</dependency>
    
```

This website uses cookies to improve your experience. We'll assume you're ok with this, but you can opt-out if

you wish. [Accept](#) [Read More](#)

Para empezar a trabajar no es necesaria ninguna acción de creación de colas, ya que el propio Sleuth crea automáticamente todo lo necesario. Al ejecutar la aplicación se encarga de gestionarlo.

Overview		Messages			Message rates			
Name	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack
sleuth.sleuth	D	idle	0	1	1	0.00/s	0.00/s	0.00/s

(<https://www.paradigmadigital.com/wp-content/uploads/2016/12/traz1.png>)

Con la anotación `@EnableZipkinStreamServer` dotas a una aplicación Spring Boot de ésta funcionalidad. Cabe destacar que existe un [api sobre el que ejecutar consultas al Zipkin Server \(http://zipkin.io/zipkin-api/#/paths/\)](http://zipkin.io/zipkin-api/#/paths/).

Las peticiones mas importantes son:

```
curl http://localhost:9411/api/v1/services
curl http://localhost:9411/api/v1/traces?serviceName=
<service-name>
curl http://localhost:9411/api/v1/trace/<id>
curl http://localhost:9411/api/v1
/dependencies?endTs=$(date +%s)
```

A continuación se muestra un ejemplo de traza en una llamada entre dos microservicios, donde `zipkin-client` y `zipkin-client-response` son los nombres de los microservicios, `994f6a06f0193d35` es el traceld y `a079a9da1fcc2960` y `1629c34056ef5226` son los correspondientes spanId. El campo `true` indica si la traza se exporta o no a Zipkin, es decir, si se envía al Zipkin Server para ser tratada, o solo aparece en los logs.

```
2016-10-03 15:41:33.983 INFO [zipkin-client,994f6a06f0193d35,a079a9da1fcc2960,true] 15004 --- [nio-8088-exec-5] c.p.m.ZipkinClientApplication : Handling home

2016-10-03 15:41:33.989 INFO [zipkin-client-response,994f6a06f0193d35,1629c34056ef5226,true] 9380 --- [nio-8089-exec-9] c.p.m.ZipkinClientReceiveApplication : GetName Method: EDU
```

This website uses cookies to improve your experience. We'll assume you're ok with this, but you can opt-out if you wish. [Accept](#) [Read More](#)

[/2016/11/td8.jpg\)](#)

De cara a mejorar el rendimiento y no sobrecargar el sistema de logs, el sistema por defecto envía un 10% de las trazas generadas al Zipkin Server. Si se desea cambiar este porcentaje, se debe modificar la siguiente propiedad:

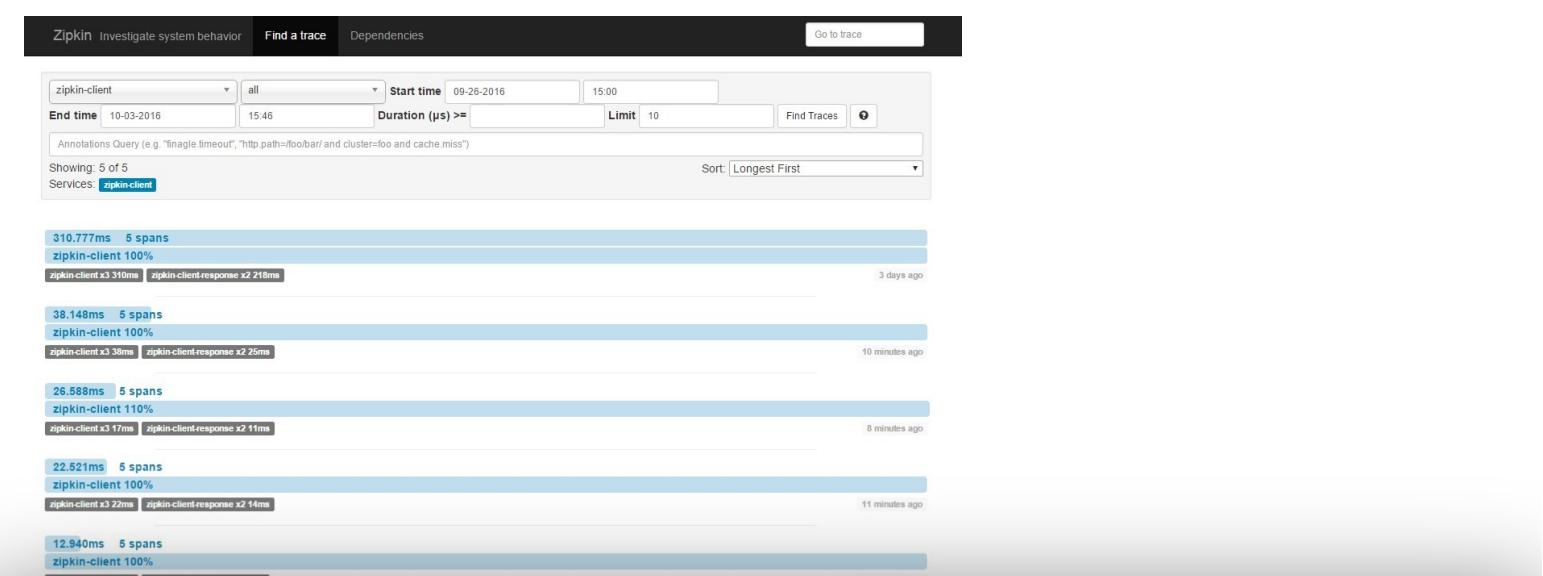
spring.sleuth.sampler.percentage = 0.2

Zipkin UI

Zipkin provee de una interfaz gráfica que agiliza las consultas a la BBDD del Zipkin Server y muestra la información recogida. Para tenerla disponible, simplemente hay que incluir la siguiente dependencia *zipkin-autoconfigure-ui* en la aplicación.

Zipkin UI permite filtrar las trazas para obtener un desgranado de las mismas. Una vez obtenido el rango de trazas que deseamos estudiar, se puede ver en detalle cada una de las **trace** y los **spans** que las componen.

Trace generados en el intervalo:



This website uses cookies to improve your experience. We'll assume you're ok with this, but you can opt-out if

you wish. [Accept](#) [Read More](#)

(<https://www.paradigmadigital.com/wp-content/uploads>)

[/2016/11/td9.jpg\)](#)

Spans de uno de los trace:



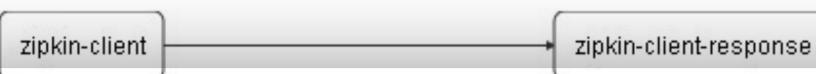
[https://www.paradigmadigital.com/wp-content/uploads/2016/11/td10.jpg"\)](https://www.paradigmadigital.com/wp-content/uploads/2016/11/td10.jpg)

Annotation (cs, sr, ss, cr) y Tags de la petición:

zipkin-client.http:/getname: 218.526ms	X		
AKA: zipkin-client,zipkin-client-response			
<hr/>			
Date	Time	Annotation	Address
30/9/2016	12:52:36	Client Send	192.168.59.3:8088 (zipkin-client)
30/9/2016	12:52:36	Server Receive	192.168.59.3:8089 (zipkin-client-response)
30/9/2016	12:52:36	Server Send	192.168.59.3:8089 (zipkin-client-response)
30/9/2016	12:52:36	Client Receive	192.168.59.3:8088 (zipkin-client)
<hr/>			
Key	Value		
http.host	localhost		
http.method	GET		
http.path	/getName		
http.url	http://localhost:8089/getName		
Server Address	192.168.59.3:8088 (zipkin-client)		

[https://www.paradigmadigital.com/wp-content/uploads/2016/11/td11.jpg"\)](https://www.paradigmadigital.com/wp-content/uploads/2016/11/td11.jpg)

Además, genera un grafo de la comunicación de los servicios en dicha traza. Haciendo clic en los elementos obtenemos información ampliada.



This website uses cookies to improve your experience. We'll assume you're ok with this, but you can opt-out if

[you wish. Accept **Read More**](#)

[/2016/11/td12.jpg](#)

A la hora de implementar una solución basada en microservicios se deben abordar ciertas peculiaridades que con aplicaciones monolíticas no ocurren, ya que al ser sistemas descentralizados el control de la comunicación entre los diferentes componentes es muy importante.

Al utilizar las herramientas Spring Cloud mencionadas no solo nos aportará la necesaria correlación de logs de una petición entre distintos microservicios, sino que también proporciona una serie de herramientas para poder visualizar la salud de nuestro ecosistema.

[Compartir en Twitter](#)[Share in Linkedin](#)

Eduardo González

Ingeniero de Software con diversos proyectos Java EE a sus espaldas. Actualmente desarrollando soluciones enfocadas a arquitecturas de microservicios con Spring Cloud y desarrollo continuo. Siempre con el objetivo de aprender y mejorar, para así poder trasladar la máxima calidad a los proyectos en los que participa.

[Ver toda la actividad de Eduardo González](#)

6 comentarios

This website uses cookies to improve your experience. We'll assume you're ok with this, but you can opt-out if you wish.

[Accept](#) [Read More](#)



Jorge Alonso

30 enero, 2017 a las 14:49

Muy interesante artículo, Eduardo, lo mejor que he visto en español sobre Sleuth y Zipkin.

En nuestra empresa hemos utilizado Sleuth y Zipkin a pequeña escala. Pero el verdadero potencial de la trazabilidad distribuida lo obtienes cuando todas las aplicaciones del sistema (o incluso de la compañía) envían información de trazabilidad, de modo que se puede hacer un seguimiento global, de principio a fin, de una petición. En ese sentido, no he encontrado ningún artículo que hable de cómo desplegar Zipkin en producción, en un sistema real, con la necesidad de gestionar un enorme volumen de peticiones. Es decir, no he localizado información de topologías de despliegue reales y a cuántas peticiones se enfrentas esas topologías. ¿Has encontrado algo al respecto?

[Responder](#)

Eduardo Gonzalez

31 enero, 2017 a las 09:36

Buenos días Jorge, muchas gracias por tu comentario.

El entorno ideal de despliegue tanto de zipkin como del resto de microservicios corporativos sería dentro de contenedores de software como puede ser Docker sobre una infraestructura Cloud ya sea pública (AWS, Google Cloud...) o privada (Openshift...). De esta manera se aprovecha la capacidad de estos sistemas en cuanto a autoescalado y aprovisionamiento sencillo de recursos para hacer frente a grandes volúmenes de peticiones.

Un saludo

[Responder](#)

Ricardo Morales

2 junio, 2017 a las 17:35

Que tal, La verdad muy bueno tu post.

Me gustaría saber si tienes documentación o algún post donde se explique la instalación y puesta en marcha en OpenShift. Muchas gracias.

This website uses cookies to improve your experience. We'll assume you're ok with this, but you can opt-out if you wish. [Accept](#) [Read More](#)

Todo el blog es genial.

[Responder](#)



Eduardo González

5 junio, 2017 a las 09:14

Hola Ricardo,

Si quieres aprender mas sobre Openshift, en los siguientes posts mi compañero Abraham lo explica muy bien:

<https://www.paradigmadigital.com/dev/openshift-enterprise-3-la-estrella-de-los-paas-12/>

(<https://www.paradigmadigital.com/dev/openshift-enterprise-3-la-estrella-de-los-paas-12/>)

<https://www.paradigmadigital.com/dev/openshift-enterprise-3-la-estrella-de-los-paas-22/>

(<https://www.paradigmadigital.com/dev/openshift-enterprise-3-la-estrella-de-los-paas-22/>)

Además, siempre es buena idea acudir a la documentación oficial.

Openshift Origin (Community): <https://docs.openshift.org/latest/welcome/index.html> (<https://docs.openshift.org/latest/welcome/index.html>)

Openshift Enterprise: <https://docs.openshift.com/container-platform/3.5/welcome/index.html>
(<https://docs.openshift.com/container-platform/3.5/welcome/index.html>)

Un saludo

[Responder](#)



john

14 septiembre, 2018 a las 13:49

es muy interesante estoy buscando un curso de netflix oss y microservices

[Responder](#)

This website uses cookies to improve your experience. We'll assume you're ok with this, but you can opt-out if

you wish. [Accept](#) [Read More](#)

Alberto

22 febrero, 2019 a las 18:41

Hola Eduardo, con respecto a la solución de utilizar sleuth me gustaría hacerte una pregunta. Si tenemos una arquitectura de microservicios donde todos y cada uno de los servicios están en alta disponibilidad, por lo que por lo menos tenemos dos instancias por cada uno, ¿como gestiona sleuth la generación de un traceld único? las soluciones que he visto siempre hablan de que el servicio que genera el traceld (el servicio que expone el API) dota de una única instancia, pero en el caso que te planteo en servicio que está cara al exterior, también está en alta disponibilidad.

[Responder](#)

ESCRIBE UN COMENTARIO

Nombre *

Mail (no será publicado) *

Página web

Comentario:

[Enviar comentario](#)[En Twitter](#)

This website uses cookies to improve your experience. We'll assume you're ok with this, but you can opt-out if you want to. Estefanía ha sido otra de las ganadoras de las jarras de 'Juego de Tronos' que hemos llevado hoy. [Accept](#) [Read More](#) [@CodemoMadrid. M...](#) <https://t.co/EPFZIJm4jH>

[En nuestro blog](#)

Serifalaris 2019, diseño creativo en estado puro

Este fin de semana tuve la suerte de poder viajar al norte, al entrañable pueblo de Getxo y asistir al evento de Serifalaris, un encuentro de diseñadores que...

{paradigma}

91 352 59 42

El responsable de los datos es
Paradigma Digital SL
Vía de las Dos Castillas, 33 - Ática 2,
28224
Pozuelo de Alarcón (Madrid)
Copyright Paradigma Digital 2019

[contacto](#)

[aviso legal y política de privacidad](#)

[cookies](#)

newsletter

Your e-mai

[Suscribirme](#)

Acepto Los Tratamientos De Datos *

Indicados En La Política De
Privacidad

Utilizaremos la información que nos facilitas para suscribirte a nuestro envío periódico de newsletters así como mandarte otras publicaciones de Paradigma que puedan ser de tu interés. Nuestra legitimación es tu consentimiento. Tus datos no se cederán a terceros, y puedes modificarlos, darte de baja o eliminarlos cuando quieras. [Aquí encontrarás toda la información sobre nuestra política de privacidad.](#)
(<https://www.paradigmadigital.com/legal/>)